```
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDDDDDDDDDDD     RRRRRRRRRRRR        IIIIIIIII     VVV         VVV  EEEEEEEEEEEEEEE  RRRRRRRRRRRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRR         RRR        III        VVV         VVV  EEE              RRR         RRR
DDD         DDD  RRRRRRRRRRRR           III        VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD         DDD  RRRRRRRRRRRR           III        VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD         DDD  RRRRRRRRRRRR           III        VVV         VVV  EEEEEEEEEEEE     RRRRRRRRRRRR
DDD         DDD  RRR   RRR              III        VVV         VVV  EEE              RRR   RRR
DDD         DDD  RRR   RRR              III        VVV         VVV  EEE              RRR   RRR
DDD         DDD  RRR   RRR              III        VVV         VVV  EEE              RRR   RRR
DDD         DDD  RRR      RRR           III           VVV   VVV     EEE              RRR      RRR
DDD         DDD  RRR      RRR           III           VVV   VVV     EEE              RRR      RRR
DDD         DDD  RRR      RRR           III           VVV   VVV     EEE              RRR      RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII           VVV        EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII           VVV        EEEEEEEEEEEEEEE  RRR         RRR
DDDDDDDDDDDD     RRR         RRR     IIIIIIIII           VVV        EEEEEEEEEEEEEEE  RRR         RRR
```

FILEID**XIDRIVER

```
XX      XX  IIIIII  DDDDDDDD    RRRRRRRR    IIIIII  VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX  IIIIII  DDDDDDDD    RRRRRRRR    IIIIII  VV      VV  EEEEEEEEEE  RRRRRRRR
XX      XX    II    DD      DD  RR      RR    II    VV      VV  EE          RR      RR
XX      XX    II    DD      DD  RR      RR    II    VV      VV  EE          RR      RR
  XX  XX      II    DD      DD  RR      RR    II    VV      VV  EE          RR      RR
  XX  XX      II    DD      DD  RR      RR    II    VV      VV  EE          RR      RR
    XX        II    DD      DD  RRRRRRRR      II    VV      VV  EEEEEEEE    RRRRRRRR
    XX        II    DD      DD  RRRRRRRR      II    VV      VV  EEEEEEEE    RRRRRRRR
  XX  XX      II    DD      DD  RR  RR        II    VV      VV  EE          RR  RR
  XX  XX      II    DD      DD  RR  RR        II    VV      VV  EE          RR  RR
XX      XX    II    DD      DD  RR    RR      II      VV  VV    EE          RR    RR    ....
XX      XX    II    DD      DD  RR    RR      II      VV  VV    EE          RR    RR    ....
XX      XX  IIIIII  DDDDDDDD    RR      RR  IIIIII      VV      EEEEEEEEEE  RR      RR  ....
XX      XX  IIIIII  DDDDDDDD    RR      RR  IIIIII      VV      EEEEEEEEEE  RR      RR  ....

MM      MM    AAAAAA    RRRRRRRR
MM      MM    AAAAAA    RRRRRRRR
MMMM  MMMM  AA      AA  RR      RR
MMMM  MMMM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM  MM  MM  AA      AA  RR      RR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AA      AA  RRRRRRRR
MM      MM  AAAAAAAAAA  RR  RR
MM      MM  AAAAAAAAAA  RR  RR
MM      MM  AA      AA  RR    RR
MM      MM  AA      AA  RR    RR
MM      MM  AA      AA  RR      RR
MM      MM  AA      AA  RR      RR
```

```
        .TITLE  XIDRIVER - VAX/VMS DMF32 PARALLEL PORT DRIVER
        .IDENT  'V04-001'
;
```



```
;
;++
; FACILITY:
;
;       VAX/VMS Executive, I/O Drivers
;
; ABSTRACT:
;
;       This driver is an example driver for the DMF32 parallel port.
;       This driver implements the DR11C compatibility mode on the device.
;       It does not implement the silo or DMA options, but serves as a
;       template to which such features could be added.
;
;       This module contains the DMF32 PARALLEL PORT driver:
;
;               Tables for loading and dispatching
;               Controller initialization routine
;               FDT routine
;               The start I/O routine
;               The interrupt service routine
;               Device specific Cancel I/O
;
; ENVIRONMENT:
;
;       Kernal Mode, Non-paged
;
; AUTHOR:
;
;       Jake VanNoy     January 1982
;
```

```
; MODIFIED BY:
;
;       V04-001 JLV0396         Jake VanNoy             6-SEP-1984
;               Add AVL to DEVCHAR.
;
;       V03-005 JLV0385         Jake VanNoy             23-JUL-1984
;               Add DPT$M_SVP to DPT.
;
;       V03-004 JLV0341         Jake VanNoy             28-MAR-1984
;               Correct Device IPL.
;
;       V03-003 WHM0002         Bill Matthews           16-Feb-1984
;               Second part of change for edit WHM0001.
;
;       V03-002 WHM0001         Bill Matthews           19-Dec-1983
;               Added code to support new IDB fields IDB$B_COMBO_VECTOR
;               and IDB$B_COMBO_CSR_OFFSET for determining the main CSR
;               address and loading the soft vector for the combo device.
;
;       V03-001 KDM0002         Kathleen D. Morse       28-Jun-1982
;               Added $DCDEF and $DYNDEF.
;
;--
```

```
        .SBTTL  Description of Interface
;++
;
; The DMF32 Parallel Port interface is a 16 bit parallel port for interfacing
; to a user device. It includes a DR11C compatibility mode (used for word
; mode within this driver), a silo (buffered) mode (not implemented by this
; driver), and a DMA mode (also not implemented by this driver). The interface
; looks like the following:
;
;
;       +--------+                          +--------+
;       |        |                          |        |
;       |   D    |---> CTRL 0 ----------->|   U    |
;       |   M    |---> CTRL 1 ----------->|   S    |
;       |   F    |                          |   E    |
;       |   3    |<--- REQ A <-----------|   R    |
;       |   2    |<--- REQ B <-----------|        |
;       |        |                          |   D    |
;       |   P    |/--- DATA  -----------\|   E    |
;       |   O    |\--- 16 LINES --------/|   V    |
;       |   R    |                          |   I    |
;       |   T    |---> New Data Ready -->|   C    |  (pulsed on write to OUTBUF)
;       |        |---> Data Tx'ed ------>|   E    |  (pulsed on read from INBUF)
;       |        |                          |        |
;       +--------+                          +--------+
;
;
;--
```

```
;++
;
; This driver may be tested using the following configuration of two DMF32's:
; The control lines (CTRL 0 and 1) should be tied into request lines (REQ A
; and B) on the other device. Setting CTRL 0 on the first device causes an
; interrupt on REQ A on the second device (provided interrupt enable A is set).
;
;       +--------+                                              +--------+
;       |        |                                              |        |
;       |   D    |-----> CTRL 0 ----------> REQ A ------>|   D    |
;       |   M    |-----> CTRL 1 ----------> REQ B ------>|   M    |
;       |   F    |                                              |   F    |
;       |   3    |<----- REQ A <---------- CTRL 0 <------|   3    |
;       |   2    |<----- REQ B <---------- CTRL 1 <------|   2    |
;       |        |                                              |        |
;       |   P    |/--- DATA  --------------------------\|   P    |
;       |   O    |\--- 16 LINES (in each direction) ----/|   O    |
;       |   R    |                                              |   R    |
;       |   T    |---> New Data Ready  (not used)               |   T    |
;       |        |---> Data Tx'ed      (not used)               |        |
;       |        |                                              |        |
;       +--------+                                              +--------+
;
;--
```

```
.SBTTL  Documentation on interface
;++
;
; The DMF32 parallel port exchanges one 16-bit word at a time.  A single
; QIO request transfers a buffer of data, with an interrupt requested for
; each word.
;
; For each buffer of data transferred, the DMF32 parallel port allows for
; the exchange of additional bits of information: the Control and Status
; Register (CSR) function (CTRL) and status (REQUEST) bits. These bits are
; accessible to an application process through the device driver QIO
; interface.  The CTRL bits are labeled CTRL 0 and CTRL 1. The REQUEST
; bits are labeled REQUEST A and REQUEST B.
;
; The user device interfaced to the DMF32 parallel port interprets the
; value of the two CTRL bits.  The QIO request that initiates the transfer
; specifies the IO$M_SETFNCT modifer to indicate a change in the value
; for the CTRL bits. The P4 argument of the request specifies this value.
; P4 bits 0 and 1 correspond to CTRL bits 0 and 1 respectively. Bits 2
; through 31 are not used. If required, the CTRL bits must be set for each
; request. The CTRL bits set in the CSR are passed directly to the user
; device.
;
; The device class for the DMF32 parallel port is DC$_REALTIME and the
; device type is DT$_XI_DR11C.  The DMF32 parallel port driver does not
; use the default buffer size field.  The value of this field is set to
; 65,535. This driver defines no device-dependent characteristics.
;
; The DMF32 parallel port can perform logical, virtual, and physical I/O
; operations. The basic I/O functions are read, write, set mode, and set
; characteristics.
;
; +----------------------------+---------------+------------------+
; ! Function Code and          !   Function    !    Function      !
; !    Arguments               !   Modifiers   !                  !
; +----------------------------+---------------+------------------+
; !                            !               !                  !
; ! IO$_READLBLK P1,P2,-       ! IO$M_SETFNCT  ! Read block      !
; !              P3,P4         ! IO$M_RESET    !                  !
; !                            ! IO$M_TIMED    !                  !
; !                            !               !                  !
; ! IO$_WRITELBLK P1,P2,-      ! IO$M_SETFNCT  !Write logical block!
; !              P3,P4         ! IO$M_RESET    !                  !
; !                            ! IO$M_TIMED    !                  !
; !                            !               !                  !
; ! IO$_SETMODE P1,P3          ! IO$M_ATTNAST  !Set PORT charact- !
; !                            !               !eristics for subse-!
; !                            !               ! quent operations !
; !                            !               !                  !
; ! IO$_SETCHAR P1,P3          ! IO$M_ATTNAST  !Set PORT charact- !
; !                            !               !eristics for subse-!
; !                            !               ! quent operations !
; !                            !               !                  !
; +----------------------------+-------+-------------+------------------+
;
; Not in above table are functions IO$_READPBLK, IO$_READVBLK, WRITEPBLK
```

```
; and WRITELBLK. There is no functional difference in these operations.
; Although the DMF32 parallel port does not differentiate between logical,
; virtual, and physical I/O functions (all are treated identically), the
; user must have the required privilege to issue a request.
;
; The function-dependent arguments for the read and write function codes are:
;
; o       P1 -- the starting virtual address of the buffer that
;         is to receive data in the case of a read operation; or, in
;         the case of a write operation, the virtual address of the
;         buffer that is to send data to the DMF32 parallel port.
;         Modify access to the buffer, rather than read or write
;         access, is checked for all block mode read and write
;         requests.
;
; o       P2 -- the size of the data buffer in bytes, that is, the
;         transfer count.  Since the DMF32 parallel port performs
;         word transfers, the transfer count must be an even value.
;
; o       P3 -- the timeout period for this request (in seconds).
;         The value specified must be equal to or greater than 2.
;         IO$M_TIMED must be specified.  The default timeout value for each
;         request is 10 seconds.
;
; o       P4 -- the value of the DMF32 parallel port Command and Status
;         Register (CSR) function (CTRL) bits to be set.  If
;         IO$M_SETFNCT is specified, the low-order three bits of P4
;         (2:0) are written to CSR CTRL bits 1:0 (respectively) at the
;         time of transfer.
;
; The transfer count specified by the P2 argument must be an even number
; of bytes.  If an odd number or more than 65534 bytes is specifed, an
; error (SS$_BADPARAM) is returned in the I/O status block (IOSB). If the
; transfer count is 0, the driver will transfer no data.  However, if
; IO$M_SETFNCT is specified and P2 is 0, the driver will set the CTRL bits
; in the DMF32 parallel port CSR, and return the current CSR status bit
; values in the IOSB.
;
; The read and write QIO functions can take three function modifiers:
;
; o       IO$M_SETFNCT - set the function (CTRL) bits in the DMF32 parallel
;         port CSR before the data transfer is initiated.  The
;         low-order two bits of the P4 argument specify the CTRL
;         bits.  The user device that interfaces the DMF32 PARALLEL
;         PORT receives the CTRL bits directly and their value is
;         interpreted entirely by the device.
;
; If an unsolicited interrupt is received from the DMF32 parallel port, no
; read or write request is posted, and the next request is for a word mode
; read, the driver will return the word read from the DMF32 parallel port
; INBUF and store it in the first word of the user's buffer. In this case
; the driver does not wait for an interrupt.
;
; o       IO$M_TIMED - set the device timeout interval for the data
;         transfer request. The P3 argument specifies the timeout
;         interval value in seconds.  For consistent results, this
```

```
;       value must be equal to or greater than 2.
;
; o     IO$M_RESET - perform a device reset to the DMF32 parallel port before
;       any I/O operation is initiated. This function does not
;       affect any other device on the system or on the DMF32.
;
; The set mode and characteristic function codes are:
;
; o     IO$_SETMODE
;
; o     IO$_SETCHAR
;
; These functions take the following device/function-dependent arguments:
;
; o     P1 - the virtual address of a quadword characteristics buffer.  If
;       the function modifer IO$M_ATTNAST is specified, P1 is the
;       address the AST service routine. In this case, if P1 is 0,
;       all attention ASTs are disabled.
;
; o     P3 - the access mode to deliver the AST (maximized with the
;       requestor's access mode).  If IO$M_ATTNAST is not specified, P3
;       is ignored.
;
;       Figure 3-4 shows the quadword P1 characteristics buffer for
;       IO$_SETMODE and IO$_SETCHAR.
;
;       31                      16 15         8 7           0
;       +-------------------------+------------+------------+
;       |                         |            |            |
;       |        not used         |    type    |   class    |
;       |                         |            |            |
;       +-------------------------+------------+------------+
;       |                                                   |
;       |               device characteristics              |
;       |                                                   |
;       +---------------------------------------------------+
;
; The IO$_SETMODE and IO$_SETCHAR function codes can take the following function
; modifier:
;
; o     IO$M_ATTNAST - enable attention AST
;
; This function modifier allows the user process to queue an attention AST
; for delivery when an asynchronous or unsolicited condition is detected
; by the DMF32 parallel port driver.  Unlike ASTs for other QIO functions,
; use of this function modifier does not increment the I/O count for the
; requesting process or lock pages in memory for I/O buffers. There must
; be an AST quota for each AST.
;
; Attention ASTs are delivered under the following conditions:
;
; o     An unsolicited interrupt from the DMF32 parallel port occurs.
;
; o     An attention AST is queued and a previous unsolicited interrupt
;       has not been acknowledged.
;
```

```
; The $CANCEL system service is used to flush attention ASTs for a specific
; channel.
;
; IO$_SETMODE!IO$M_ATTNAST and IO$_SETCHAR!IO$M_ATTNAST are one-time AST
; enables; they must be explicitly re-enabled once the AST has been
; delivered if the user desires notification of the next interrupt.  Use
; of this function modifier does not update the device characteristics.
;
; After the AST is delivered, the QIO astprm parameter contains the
; contents of the DMF32 parallel port CSR in the low two bytes and the
; value read from the DMF32 parallel port INBUF in the high two bytes.
;
; On completion of each read or write request, the I/O status block
; is filled with system and DMF32 parallel port status information.
;
;                               +2                      IOSB
;       +-----------------------+-----------------------+
;       |                       |                       |
;       |      byte count       |        status         |
;       |                       |                       |
;       +-----------------------+-----------------------+
;       |                       |                       |
;       |      unused           |       PORT CSR        |
;       |                       |                       |
;       +-----------------------+-----------------------+
;
;
;--
```

```
        .SBTTL  External and local symbol definitions

; External symbols

        $ACBDEF                         ; AST control block
        $CRBDEF                         ; Channel request block
        $DCDEF                          ; Device types
        $DDBDEF                         ; Device data block
        $DPTDEF                         ; Driver prolog table
        $DYNDEF                         ; Dynamic data structure types
        $IDBDEF                         ; Interrupt data block
        $IODEF                          ; I/O function codes
        $IPLDEF                         ; Hardware IPL definitions
        $IRPDEF                         ; I/O request packet
        $PRDEF                          ; Internal processor registers
        $PRIDEF                         ; Scheduler priority increments
        $SSDEF                          ; System messages
        $UCBDEF                         ; Unit control block
        $VECDEF                         ; Interrupt vector block

; Local symbols

; Argument list (AP) offsets for device-dependent QIO parameters

P1      = 0                             ; First QIO parameter
P2      = 4                             ; Second QIO parameter
P3      = 8                             ; Third QIO parameter
P4      = 12                            ; Fourth QIO parameter
P5      = 16                            ; Fifth QIO parameter
P6      = 20                            ; Sixth QIO parameter

; Other constants

XI_DEF_TIMEOUT  = 10                    ; 10 second default device timeout
XI_DEF_BUFSIZ   = 65535                 ; Default buffer size
XI$K_VEC_OFFSET = 2                     ; Vector offset

;
; Macros
;

;
; The SETCTRL macro sets the CTRL 0 and 1 lines as they have been
; specified in P4 in a read or write QIO. They are cleared and a wait
; occurs before being set. This is because testing for this example
; driver was done between two DMF32's in word mode, and the delay is so the
; microcode on the DMF32 can see the control line changes.
;

.MACRO  SETCTRL
        BICW    #XI_CSR$M_CTRL0!XI_CSR$M_CTRL1,XI_CSR(R4)
        CLRL    -(SP)
        TIMEWAIT -
                TIME = #2,-
                BITVAL = #1,-
```

```
                        SOURCE = (SP),-
                        CONTEXT = L,-
                        SENSE = .TRUE.
                TSTL    (SP)+
                BISW    IRP$L_SEGVBN(R3),XI_CSR(R4)
.ENDM           SETCTRL
```

```
; UCB_XI definitions that follow the standard UCB fields
        $DEFINI UCB
        .=UCB$L_DPC+4   ;
$DEF    UCB$L_XI_ATTN
                        .BLKL   1       ; Attention AST queue
$DEF    UCB$L_XI_DPR
                        .BLKL   1       ; Word count?
$DEF    UCB$W_XI_INBUF
                        .BLKW   1       ; Input buffer temporary
$DEF    UCB$W_XI_CSR
                        .BLKW   1       ; CSR temporary
; Bit positions for device-dependent status field in UCB (UCB$W_DEVSTS)
        $VIELD  UCB,0,<-                ; UCB device specific bit definitions
                <ATTNAST,,M>,-
                <UNEXPT,,M>-
                >
UCB$K_SIZE=.
        $DEFEND UCB
```

```
;
; DMF32 Parallel Port CSR definitions
;
        $DEFINI XI

$DEF    XI_CSR                          ; Device CSR

; Bit positions for device control/status register

        $VIELD  XI_CSR,0,<-             ; Control/status register
                <CTRL0,,M>,-            ; Control line 0
                <CTRL1,,M>,-            ; Control line 1
                <NPR_PS,,M>,-           ; NPR Primary/Secondary
                <INDREG,2,M>,-          ; Indirect Register Address
                <INTENB_A,,M>,-         ; Interrupt Enable A
                <INTENB_B,,M>,-         ; Interrupt Enable B
                <REQ_A,,M>,-            ; Request A
                <DONE_P,,M>,-           ; Done Primary
                <DONE_S,,M>,-           ; Done Secondary
                <,,M>,-                 ; unused
                <FLUSH,,M>,-            ; Flush Buffer
                <,,M>,-                 ; unused
                <NXMERR,,M>,-           ; Non-existent memory error
                <RESET,,M>,-            ; Master Reset
                <REQ_B,,M>-             ; Request B
        >

XI_CSR$M_IEAB   = <XI_CSR$M_INTENB_A>!<XI_CSR$M_INTENB_B> ; Interrupt enable mask

                        .BLKW   1
$DEF    XI_OUTBUF                       ; Output buffer Register
                        .BLKW   1

; Note that XI_INBUF and XI_MISC are at the same offset

$DEF    XI_INBUF                        ; Input buffer Register (when read)
$DEF    XI_MISC                         ; Miscellaneous Register (when written)

; Bit positions for miscellaneous register

        $VIELD  XI_MISC,0,<-            ; Miscellaneous register
                <MODE,4,M>,-            ; Mode
                <,10,M>,-               ; unused
                <SECBUF,,M>-            ; Secondary Buffer Address, Bit 17
                <PRIBUF,,M>-            ; Primary Buffer Address, Bit 17
        >
                        .BLKW   1
$DEF    XI_IND                          ; Indirect Register
                        .BLKW   1

        $DEFEND XI                      ; End of PORT CSR definitions
```

```
        .SBTTL  Device Driver Tables

; Driver prologue table

        DPTAB   -                               ; DPT-creation macro
                END=XI_END,-                    ; End of driver label
                ADAPTER=UBA,-                   ; Adapter type
                FLAGS=DPT$M_SVP,-               ; Allocate system page table
                UCBSIZE=UCB$K_SIZE,-            ; UCB size
                NAME=XIDRIVER                   ; Driver name

        DPT_STORE INIT                          ; Start of load
                                                ; initialization table
        DPT_STORE UCB,UCB$B_FIPL,B,8            ; Device fork IPL
        DPT_STORE UCB,UCB$B_DIPL,B,21           ; Device interrupt IPL
        DPT_STORE UCB,UCB$L_DEVCHAR,L,<-        ; Device characteristics
                DEV$M_AVL!-                     ; Available
                DEV$M_RTM!-                     ; Real Time device
                DEV$M_IDV!-                     ;   input device
                DEV$M_ODV>                      ;   output device
        DPT_STORE UCB,UCB$B_DEVCLASS,B,DC$_REALTIME     ; Device class
        DPT_STORE UCB,UCB$B_DEVTYPE,B,DT$_XI_DR11C      ; Device Type
        DPT_STORE UCB,UCB$W_DEVBUFSIZ,W,-       ; Default buffer size
                XI_DEF_BUFSIZ
        DPT_STORE REINIT                        ; Start of reload
                                                ; initialization table
        DPT_STORE DDB,DDB$L_DDT,D,XI$DDT        ; Address of DDT
        DPT_STORE CRB,CRB$L_INTD+4,D,-          ; Address of interrupt
                XI_INTERRUPT                    ; service routine
        DPT_STORE CRB,CRB$L_INTD2+4,D,-         ; Address of interrupt
                XI_INTERRUPT                    ; service routine
        DPT_STORE CRB,CRB$L_INTD+VEC$L_INITIAL,-; Address of controller
                D,XI_CONTROL_INIT               ; initialization routine
        DPT_STORE END                           ; End of initialization
                                                ; tables

; Driver dispatch table

        DDTAB   -                               ; DDT-creation macro
                DEVNAM=XI,-                     ; Name of device
                START=XI_START,-                ; Start I/O routine
                FUNCTB=XI_FUNCTABLE,-           ; FDT address
                CANCEL=XI_CANCEL                ; Cancel I/O routine
.PAGE
;
; Function dispatch table
;
XI_FUNCTABLE:                                   ; FDT for driver

        ; Valid I/O functions

        FUNCTAB ,-
                <READPBLK,READLBLK,READVBLK, -
                WRITEPBLK,WRITELBLK,WRITEVBLK,-
                SETMODE,SETCHAR,SENSEMODE,SENSECHAR>
```

```
        FUNCTAB ,                       ; No buffered functions

        FUNCTAB XI_READ_WRITE,-                 ; Device-specific FDT
                <READPBLK,READLBLK,READVBLK, -
                WRITEPBLK,WRITELBLK,WRITEVBLK>
        FUNCTAB +EXE$READ,<READPBLK,READLBLK,READVBLK>
        FUNCTAB +EXE$WRITE,<WRITEPBLK,WRITELBLK,WRITEVBLK>
        FUNCTAB XI_SETMODE,<SETMODE,SETCHAR>
        FUNCTAB +EXE$SENSEMODE,<SENSEMODE,SENSECHAR>
```

```
        .SBTTL  XI_CONTROL_INIT, Controller initialization

;++
; XI_CONTROL_INIT, Called when driver is loaded, system is booted, or
; power failure recovery.
;
; Functional Description:
;
;       1) Allocates the direct data path permanently
;       2) Assigns the controller data channel permanently
;       3) Clears the Control and Status Register
;       4) If power recovery, requests device time-out
;
; Inputs:
;
;       R4 = address of CSR
;       R5 = address of IDB
;       R6 = address of DDB
;       R8 = address of CRB
;
; Outputs:
;
;       VEC$V_PATHLOCK bit set in CRB$L_INTD+VEC$B_DATAPATH
;       UCB address placed into IDB$L_OWNER
;
;
;--

XI_CONTROL_INIT:

        MOVL    IDB$L_UCBLST(R5),R0     ; Address of UCB
        MOVL    R0,IDB$L_OWNER(R5)      ; Make permanent controller owner
        BISW    #UCB$M_ONLINE, -
                UCB$W_STS(R0)           ; Set device status "on-line"

; If powerfail has occured and device was active, force device time-out.
; The user can set his own time-out interval for each request.  Time-
; out is forced so a very long time-out period will be short circuited.

        BBS     #UCB$V_POWER, -
                UCB$W_STS(R0),10$       ; Branch if powerfail
        BISB    #VEC$M_PATHLOCK, -
                CRB$L_INTD+VEC$B_DATAPATH(R8) ; Permanently allocate direct datapath
10$:

        CVTBL   IDB$B_COMBO_CSR_OFFSET(R5),R0   ; GET OFFSET TO MAIN DMF CSR
        SUBB3   IDB$B_COMBO_VECTOR_OFFSET(R5),- ; CALCULATE AND LOAD THE
                IDB$B_VECTOR(R5),(R4)[R0]       ;       VECTOR ADDRESS
        BSBW    XI_DEV_RESET            ; Reset port
        RSB                             ; Done
```

```
        .SBTTL  XI_READ_WRITE,   Data transfer FDT

;++
; XI_READ_WRITE, FDT for READLBLK,READVBLK,READPBLK,WRITELBLK,WRITEVBLK,
;                        WRITEPBLK
;
; Functional description:
;
;       1) Rejects QUEUE I/O's with odd transfer count
;       2) Rejects QUEUE I/O's for DMA request to UBA Direct Data
;          PATH on odd byte boundary
;       3) Stores request time-out count specified in P3 into IRP
;       4) Stores CTRL bits specified in P4 into IRP
;       6) Checks block mode transfers for memory modify access
;
; Inputs:
;
;       R3 = Address of IRP
;       R4 = Address of PCB
;       R5 = Address of UCB
;       R6 = Address of CCB
;       AP = Address of P1
;               P1 = Buffer Address
;               P2 = Buffer size in bytes
;               P3 = Request time-out period (conditional on IO$M_TIMED)
;               P4 = Value for CSR CTRL bits (conditional on IO$M_SETFNCT)
;               P5 = 0 for Request A, 1 for Request B (DMA)
;
; Outputs:
;
;       R0 = Error status if odd transfer count
;       IRP$L_MEDIA = Time-out count for this request
;       IRP$L_SEGVBN = CTRL bits for PORT CSR
;
;--

XI_READ_WRITE:

        BLBC    P2(AP),20$              ; Branch if transfer count even
10$:    MOVZWL  #SS$_BADPARAM,R0        ; Set error status code
        JMP     G^EXE$ABORTIO           ; Abort request

20$:    MOVZWL  IRP$W_FUNC(R3),R1       ; Fetch I/O function code
        MOVL    P3(AP),IRP$L_MEDIA(R3)  ; Set request specific time-out count
        BBS     #IO$V_TIMED,R1,30$      ; Branch if time-out specified
        MOVZWL  #XI_DEF_TIMEOUT, -
                IRP$L_MEDIA(R3)         ; Else set default timeout value
30$:    EXTZV   #0,#2,P4(AP),-
                IRP$L_SEGVBN(R3)        ; Get value for CTRL bits
        RSB                             ; Return
```

```
        .SBTTL  XI_SETMODE,      Set Mode, Set Char FDT

;++
; XI_SETMODE, FDT routine to process SET MODE and SET CHARACTERISTICS
;
; Functional description:
;
;       If IO$M_ATTNAST modifier is set, queue attention AST for device
;       Else, finish I/O.
;
; Inputs:
;
;       R3 = I/O packet address
;       R4 = PCB address
;       R5 = UCB address
;       R6 = CCB address
;       R7 = Function code
;       AP = QIO Paramater list address
;
; Outputs:
;
;       If IO$M_ATTNAST is specified, queue AST on UCB attention AST list.
;       Else, use exec routine to update device characteristics
;
;--
XI_SETMODE:

        MOVZWL  IRP$W_FUNC(R3),R0       ; Get entire function code
        BBC     #IO$V_ATTNAST,R0,20$    ; Branch if not an ATTN AST

; Attention AST request

        PUSHR   #^M<R4,R7>
        MOVAB   UCB$L_XI_ATTN(R5),R7    ; Address of ATTN AST control block list
        JSB     G^COM$SETATTNAST        ; Set up attention AST
        POPR    #^M<R4,R7>
        BLBC    R0,30$                  ; Branch if error
        BISW    #UCB$M_ATTNAST, -
                UCB$W_DEVSTS(R5)        ; Flag ATTN AST expected.
        BBC     #UCB$V_UNEXPT, -
                UCB$W_DEVSTS(R5),10$    ; Deliver AST if unsolicited interrupt
        BSBW    XI_DEL_ATTNAST
10$:    JMP     G^EXE$FINISHIO          ; Thats all for now

20$:    JMP     G^EXE$SETCHAR           ; Set device characteristics

30$:    CLRL    R1                      ; zero R1
        JMP     G^EXE$ABORTIO           ; Abort I/O with R0 as status
```

```
        .SBTTL  XI_START,       Start I/O routines
;++
; XI_START - Start a data transfer, set characteristics, enable ATTN AST.
;
; Functional Description:
;
;       This routine has one major function:
;
;       1) Start an I/O transfer. The CTRL bits in the port CSR are set.  If
;          the transfer count is zero, the STATUS bits in the PORT CSR
;          are read and the request completed.
;
; Inputs:
;
;       R3 = Address of the I/O request packet
;       R5 = Address of the UCB
;
; Outputs:
;
;       R0 = final status and number of bytes transferred
;       R1 = value of CSR STATUS bits
;
;--

XI_START:

; Retrieve the address of the device CSR

        ASSUME  IDB$L_CSR EQ 0
        MOVL    UCB$L_CRB(R5),R4        ; Address of CRB
        MOVL    @CRB$L_INTD+VEC$L_IDB(R4),R4
                                        ; Address of CSR

; Fetch the I/O function code

        MOVZWL  IRP$W_FUNC(R3),R1       ; Get entire function code
        MOVW    R1,UCB$W_FUNC(R5)       ; Save FUNC in UCB
        EXTZV   #IO$V_FCODE, -
                #IO$S_FCODE,R1,R2       ; Extract function field

; If subfunction modifier for device reset is set, do one here

        BBC     S^#IO$V_RESET,R1,40$    ; Branch if not device reset
        BSBW    XI_DEV_RESET            ; Reset port

; This must be a data transfer function - i.e. READ OR WRITE
; Check to see if this is a zero length transfer.
; If so, only set CSR CTRL bits and return STATUS from CSR

40$:    TSTW    UCB$W_BCNT(R5)          ; Is transfer count zero?
        BNEQ    100$                    ; No, continue with data transfer
        BBC     S^#IO$V_SETFNCT,R1,60$  ; Set CSR CTRL specified?
        DSBINT                          ; Disable Interrupts
        SETCTRL                         ; Set CTRL bits in CSR
        MOVZWL  XI_CSR(R4),R1           ; Save CSR
        ENBINT                          ; Enable Interrupts
```

```
        BRB     70$                             ; Skip clearing of R1

60$:    CLRL    R1                              ; Clear R1
70$:    BISW    #XI_CSR$M_IEAB,-
                XI_CSR(R4)                      ; Enable device interrupts (A & B)
        MOVZWL  #SS$_NORMAL,R0                  ; Set success
        REQCOM                                  ; Request done

;
; Do the read or the write
;

100$:
        MOVZWL  UCB$W_BCNT(R5),R0               ; Get byte count
        ASHL    #-1,R0,UCB$L_XI_DPR(R5)         ; Make byte count into word count

        .SBTTL  - word mode tranfer
;++
; WORD MODE -- Process word mode (interrupt per word) transfer
;
; FUNCTIONAL DESCRIPTION:
;
;       Data is transferred one word at a time with an interrupt for each word.
;       The request is handled separately for a write (from memory to port
;       and a read (from port to memory).
;       For a write, data is fetched from memory, loaded into the ODR of the
;       port and the system waits for an interrupt.  For a read, the system
;       waits for a port interrupt and the INBUF is transferred into memory.
;       If the unsolicited interrupt flag is set, the first word is transferred
;       directly into memory withou waiting for an interrupt.
;--

WORD_MODE:

; Dispatch to separate loops on READ or WRITE

10$:
        CMPB    #IO$_READPBLK,R2                ; Check for read function
        BEQL    WORD_MODE_READ
.PAGE
;++
; WORD MODE WRITE -- Write (output) in word mode
;
; FUNCTIONAL DESCRIPTION:
;
;       Transfer the requested number of words from user memory to
;       the port OUTBUF one word at a time, wait for interrupt for each
;       word.
;--

WORD_MODE_WRITE:
10$:
        BSBW    MOVFRUSER                       ; Get two bytes from user buffer
        DSBINT                                  ; Lock out interrupts
                                                ; Flag interrupt expected
        MOVW    R1,XI_OUTBUF(R4)                ; Move data to port
```

```
        BISW    #XI_CSR$M_IEAB, -
                XI_CSR(R4)                   ; Set Interrupt Enable (A & B)
        SETCTRL                              ; Clear and set CTRL bits

; Wait for interrupt, powerfail, or device time-out

        WFIKPCH XI_TIME_OUTW,IRP$L_MEDIA(R3)

; Decrement transfer count, and loop until done

        IOFORK                               ; Fork to lower IPL
        DECW    UCB$L_XI_DPR(R5)             ; All words transferred?
        BNEQ    10$                          ; No, loop until finished.

; Transfer is done, clear interrupt expected flag and FORK
; All words read or written in WORD MODE.  Finish I/O.

RETURN_STATUS:

        MOVZWL  #SS$_NORMAL,R0               ; Complete success status
        MULW3   #2,UCB$L_XI_DPR(R5),R1       ; Calculate actual bytes xfered
        SUBW3   R1,UCB$W_BCNT(R5),R1         ; From requested number of bytes
        INSV    R1,#16,#16,R0                ; And place in high word of R0
        MOVZWL  UCB$W_XI_CSR(R5),R1          ; Return CSR status
        BICW    #<XI_CSR$M_CTRL0! -
                  XI_CSR$M_CTRL1>,-
                XI_CSR(R4)                   ; Clear CTRL bits
        BISW    #XI_CSR$M_IEAB,-
                XI_CSR(R4)                   ; Enable device interrupts (A & B)
        REQCOM                               ; Finish request in exec
.PAGE
;++
; WORD MODE READ -- Read (input) in word mode
;
; FUNCTIONAL DESCRIPTION:
;
;       Transfer the requested number of words from the port INBUF into
;       user memory one word at a time, wait for interrupt for each word.
;       If the unexpected (unsolicited) interrupt bit is set, transfer the
;       first (last received) word to memory without waiting for an
;       interrupt.
;--

WORD_MODE_READ:
        SETIPL  UCB$B_DIPL(R5)               ; Lock out interrupts

; If an unexpected (unsolicited) interrupt has occured, assume it
; is for this READ request and return value to user buffer without
; waiting for an interrupt.

        BBSC    #UCB$V_UNEXPT, -
                UCB$W_DEVSTS(R5),20$         ; Branch if unexpected interrupt
        DSBINT
10$:    BISW    #XI_CSR$M_IEAB, -
                XI_CSR(R4)                   ; Set Interrupt Enable (A & B)
        SETCTRL                              ; Clear and set CTRL bits
```

```
; Wait for interrupt, powerfail, or device time-out

        WFIKPCH XI_TIME_OUTW,IRP$L_MEDIA(R3)

; Decrement transfer count, and loop until done

        IOFORK                          ; Fork to lower IPL
20$:
        BSBW    MOVTOUSER               ; Store two bytes into user buffer

; Send interrupt back to sender.  Acknowledge we got last word.

        DSBINT
        DECW    UCB$L_XI_DPR(R5)        ; Decrement transfer count
        BNEQ    10$                     ; Loop until all words transferred
        SETCTRL
        ENBINT
        BRW     RETURN_STATUS           ; Finish request in common code
.PAGE
;
; MOVFRUSER - Routine to fetch two bytes from user buffer.
;
; INPUTS:
;
;       R5 = UCB address
;
; OUTPUTS:
;
;       R1 = Two bytes of data from users buffer
;       Buffer descriptor in UCB is updated.
;
        .ENABL  LSB
MOVFRUSER:
        MOVAL   -(SP),R1                ; Address of temporary stack loc
        MOVZBL  #2,R2                   ; Fetch two bytes
        JSB     G^IOC$MOVFRUSER         ; Call exec routine to do the deed
        MOVL    (SP)+,R1                ; Retrieve the bytes
        BRB     20$                     ; Update UCB buffer pointers

;
; MOVTOUSER - Routine to store two bytes into users buffer.
;
; INPUTS:
;
;       R5 = UCB address
;       UCB$W_XI_INBUF(R5) = Location where two bytes are saved
;
; OUTPUTS:
;
;       Two bytes are stored in user buffer and buffer descriptor in
;       UCB is updated.
;

MOVTOUSER:
        MOVAB   UCB$W_XI_INBUF(R5),R1   ; Address of internal buffer
```

```
        MOVZBL  #2,R2
        JSB     G^IOC$MOVTOUSER          ; Call exec
20$:                                     ; Update buffer pointers in UCB
        ADDW    #2,UCB$W_BOFF(R5)        ; Add two to buffer descriptor
        BICW    #^C<^X01FF>,UCB$W_BOFF(R5) ; Modulo the page size
        BNEQ    30$                      ; If NEQ, no page boundary crossed
        ADDL    #4,UCB$L_SVAPTE(R5)      ; Point to next page
30$:
        RSB
        .DSABL  LSB
```

```
        .SBTTL  XI_TIME_OUTW,  Device time-out routine
;++
; Device TIME-OUT
;
; Clear port CSR
; Return error status
;
; Power failure will appear as a device time-out
;--

XI_TIME_OUTW:                           ; Time-out for WORD mode transfer

        BSBW    XI_DEV_RESET            ; Reset controller
        MOVZWL  #SS$_TIMEOUT,R0         ; Error status
        CLRL    R1
        CLRW    UCB$W_DEVSTS(R5)        ; Clear ATTN AST flags
        BICW    #<UCB$M_TIM     ! -
                  UCB$M_INT     ! -
                  UCB$M_TIMOUT  ! -
                  UCB$M_CANCEL  ! -
                  UCB$M_POWER>,-
                UCB$W_STS(R5)           ; Clear unit status flags
        REQCOM                          ; Complete I/O in exec
```

```
        .SBTTL  XI_INTERRUPT,    Interrupt service routine for PORT
;++
; XI_INTERRUPT, Handles interrupts generated by port
;
; Functional description:
;
;       This routine is entered whenever an interrupt is generated
;       by the port.  It checks that an interrupt was expected.
;       If not, it sets the unexpected (unsolicited) interrupt flag.
;       All device registers are read and stored into the UCB.
;       If an interrupt was expected, it calls the driver back at its Wait
;       For Interrupt point.
;       Deliver ATTN AST's if unexpected interrupt.
;
; Inputs:
;
;       00(SP) = Pointer to address of the device IDB
;       04(SP) = saved R0
;       08(SP) = saved R1
;       12(SP) = saved R2
;       16(SP) = saved R3
;       20(SP) = saved R4
;       24(SP) = saved R5
;       28(SP) = saved PSL
;       32(SP) = saved PC
;
; Outputs:
;
;       The driver is called at its Wait For Interrupt point if an
;       interrupt was expected.
;       The current value of the port CSR's are stored in the UCB.
;
;--
XI_INTERRUPT:                                   ; Interrupt service for PORT

        MOVL    @(SP)+,R4                       ; Address of IDB and pop SP
        MOVQ    (R4),R4                         ; CSR and UCB address from IDB
;
; Read INBUF and CSR
;
        MOVW    XI_INBUF(R4), -
                UCB$W_XI_INBUF(R5)              ; Read input data
        MOVW    XI_CSR(R4), -
                UCB$W_XI_CSR(R5)                ; Read CSR

; Check to see if device transfer request active or not
; If so, call driver back at Wait for Interrupt point and
; Clear unexpected interrupt flag.

        BBCC    #UCB$V_INT, -
                UCB$W_STS(R5),10$               ; If clear, no interrupt expected

; Interrupt expected, clear unexpected interrupt flag and call driver
; back.

        BICW    #UCB$M_UNEXPT, -
```

```
                UCB$W_DEVSTS(R5)        ; Clear unexpected interrupt flag
        MOVL    UCB$L_FR3(R5),R3        ; Restore drivers R3
        JSB     @UCB$L_FPC(R5)          ; Call driver back after WFIKPCH
        BRB     20$                     ; Exit

; Deliver ATTN AST's if no interrupt expected and set unexpected
; interrupt flag.

10$:
        BISW    #UCB$M_UNEXPT, -
                UCB$W_DEVSTS(R5)        ; Set unexpected interrupt flag
        BSBW    XI_DEL_ATTNAST          ; Deliver ATTN AST's
        BISW    #XI_CSR$M_IEAB,-
                XI_CSR(R4)              ; Enable device interrupts (A & B)

; Restore registers and return from interrupt

20$:
        POPR    #^M<R0,R1,R2,R3,R4,R5>  ; Restore registers
        REI                             ; Return from interrupt
```

```
        .SBTTL  XI_CANCEL,      Cancel I/O routine
;++
; XI_CANCEL, Cancels an I/O operation in progress
;
; Functional description:
;
;       Flushes Attention AST queue for the user.
;       If transfer in progress, do a device reset to port
;       and finish the request.
;       Clear interrupt expected flag.
;
; Inputs:
;
;       R2 = negated value of channel index
;       R3 = address of current IRP
;       R4 = address of the PCB requesting the cancel
;       R5 = address of the device's UCB
;
; Outputs:
;
;--
XI_CANCEL:                                      ; Cancel I/O

        BBCC    #UCB$V_ATTNAST, -
                UCB$W_DEVSTS(R5),20$    ; ATTN AST enabled?

; Finish all ATTN AST's for this process.

        PUSHR   #^M<R2,R6,R7>
        MOVL    R2,R6                   ; Set up channel number
        MOVAB   UCB$L_XI_ATTN(R5),R7    ; Address of listhead
        JSB     G^COM$FLUSHATTNS        ; Flush ATTN AST's for process
        POPR    #^M<R2,R6,R7>
        BICW    #UCB$M_UNEXPT, -
                UCB$W_DEVSTS(R5)        ; Clear unexpected interrupt flag

; Check to see if a data transfer request is in progress
; for this process on this channel

20$:
        SETIPL  UCB$B_DIPL(R5)          ; Lock out device interrupts
        JSB     G^IOC$CANCELIO          ; Check if transfer going
        BBC     #UCB$V_CANCEL, -
                UCB$W_STS(R5),30$       ; Branch if not for this guy

        MOVZWL  #SS$_CANCEL,R0          ; Status is request canceled
        CLRL    R1
        CLRW    UCB$W_DEVSTS(R5)        ; Clear unexpected interrupt flag
        BICW    #<UCB$M_TIM     ! -
                  UCB$M_BSY     ! -
                  UCB$M_CANCEL  ! -
                  UCB$M_INT     ! -
                  UCB$M_TIMOUT>,-
                UCB$W_STS(R5)           ; Clear unit status flags
        REQCOM                          ; Jump to exec to finish I/O
```

```
30$:
        SETIPL  UCB$B_FIPL(R5)          ; Lower to FORK IPL
        RSB                             ; Return
```

```
        .SBTTL  XI_DEL_ATTNAST,  Deliver ATTN AST's
;++
; XI_DEL_ATTNAST, Deliver all outstanding ATTN AST's
;
; Functional description:
;
;       This routine is used by the port driver to deliver all of the
;       outstanding attention AST's.  It is copied from COM$DELATTNAST in
;       the exec.  In addition, it places the saved value of the port CSR
;       and Input Data Buffer Register in the AST paramater.
;
; Inputs:
;
;       R5 = UCB of unit
;
; Outputs:
;
;       R0,R1,R2 Destroyed
;       R3,R4,R5 Preserved
;--
XI_DEL_ATTNAST:
        BBCC    #UCB$V_ATTNAST, -
                UCB$W_DEVSTS(R5),30$    ; Any ATTN AST's expected?
        PUSHR   #^M<R3,R4,R5>           ; Save R3,R4,R5
10$:    MOVL    8(SP),R1                ; Get address of UCB
        MOVAB   UCB$L_XI_ATTN(R1),R2    ; Address of ATTN AST listhead
        MOVL    (R2),R5                 ; Address of next entry on list
        BEQL    20$                     ; No next entry, end of loop
        BICW    #UCB$M_UNEXPT, -
                UCB$W_DEVSTS(R1)        ; Clear unexpected interrupt flag
        MOVL    (R5),(R2)               ; Close list
        MOVW    UCB$W_XI_INBUF(R1), -
                ACB$L_KAST+6(R5)        ; Store INBUF in AST paramater
        MOVW    UCB$W_XI_CSR(R1), -
                ACB$L_KAST+4(R5)        ; Store CSR in AST paramater
        PUSHAB  B^10$                   ; Set return address for FORK
                                        ;   so that it loops through all AST's
        FORK                            ; FORK for this AST

; AST fork procedure

        MOVQ    ACB$L_KAST(R5),ACB$L_AST(R5)
                                        ; Re-arrange entries
        MOVB    ACB$L_KAST+8(R5),ACB$B_RMOD(R5)
        MOVL    ACB$L_KAST+12(R5),ACB$L_PID(R5)
        CLRL    ACB$L_KAST(R5)
        MOVZBL  #PRI$_IOCOM,R2          ; Set up priority increment
        JMP     G^SCH$QAST              ; Queue the AST

20$:    POPR    #^M<R3,R4,R5>           ; Restore registers
30$:    RSB                             ; Return
```

```
        .SBTTL  XI_DEV_RESET,    Device reset routine
;++
; XI_DEV_RESET - Device reset routine
;
; This routine raises IPL to device IPL, performs a device reset to
; the required controler, and re-enables device interrupts.
;
; Inputs:
;
;       R4 - Address of Control and Status Register
;       R5 - Address of UCB
;
; Outputs:
;
;       Controller is reset, controller interrupts are enabled
;
;--

XI_DEV_RESET:

        DSBINT                          ; Raise IPL to lock all interrupts

        BISW    #XI_CSR$M_RESET,-
                XI_CSR(R4)              ; Reset device

        TIMEWAIT -                      ; Timewait to allow reset
                TIME = #500,-
                BITVAL = #XI_CSR$M_RESET,-
                SOURCE = XI_CSR(R4),-
                CONTEXT = W,-
                SENSE = .FALSE.

        BISW    #XI_CSR$M_IEAB,-
                XI_CSR(R4)              ; Enable device interrupts (A & B)

        ENBINT                          ; Restore IPL
        RSB

XI_END:                                 ; End of driver label
        .END
```

0107 AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY

XADRIVER
MAR

CONINTERR
LIS

XIDRIVER
MAR

PAMAC
MAR

CNDRIVER
LIS